Aims

This exercise aims to get you to:

- Compile, run, and debug MapReduce tasks via Command Line
- Compile, run, and debug MapReduce tasks via Eclipse

One Tip on Hadoop File System Shell

Following are the three commands which appear same but have minute differences:

```
1. hadoop fs {args}

2. hadoop dfs {args}

3. hdfs dfs {args}
```

The first command: fs relates to a generic file system which can point to any file systems like local, HDFS etc. So this can be used when you are dealing with different file systems such as Local FS, HFTP FS, S3 FS, and others.

The second command: dfs is very specific to HDFS. It would work for operation relates to HDFS. This has been deprecated and we should use hdfs dfs instead.

The third command: It is the same as 2nd. It would work for all the operations related to HDFS and is the recommended command instead of hadoop dfs.

Therefore, when dealing with HDFS in our labs, it is always recommended to use **`hdfs dfs {args}`**.

Compile and Run "WordCount" via Command Line

This exercise aims to make you know how to compile your MapReduce java program and how to run it in Hadoop.

1. Download the sample code "WordCount.java":

```
$ wget http://www.cse.unsw.edu.au/~z3515164/WordCount.java
```

2. Add the following environment variables to the end of file ~/.bashrc:

```
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Save the file, and then run the following command to take these configurations into effect:

```
$ source ~/.bashrc
```

3. Compile WordCount.java and create a jar:

```
$ $HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main WordCount.java
$ jar cf wc.jar WordCount*.class
```

4. Generate two files, file1 and file2 in folder TestFiles at your home folder:

```
$ mkdir ~/TestFiles
$ echo Hello World Bye World > ~/TestFiles/file1
$ echo Hello Hadoop Goodbye Hadoop > ~/TestFiles/file2
```

5. Start HDFS and YARN, and put the two files to HDFS:

```
$ $HADOOP_HOME/sbin/start-all.sh
$ $HADOOP_HOME/bin/hdfs dfs –mkdir input
$ $HADOOP_HOME/bin/hdfs dfs –put ~/TestFiles/* input
```

6. Run the application:

```
$ $HADOOP_HOME/bin/hadoop jar wc.jar WordCount input output
```

7. Check out the output:

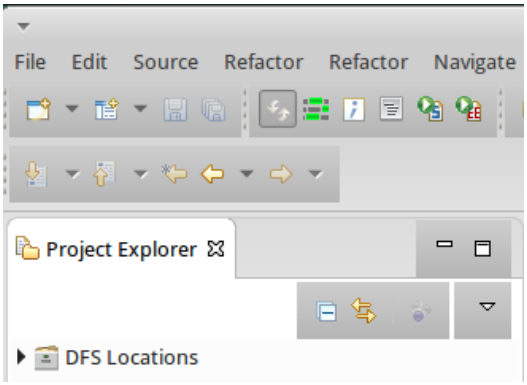
```
$ $HADOOP_HOME/bin/hdfs dfs –cat output/*
```

Create a WordCount Project in Eclipse

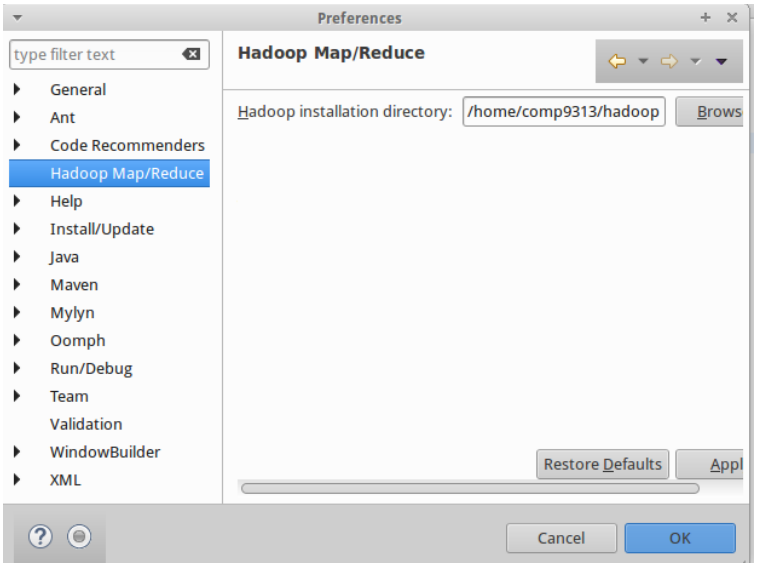
Eclipse Juno (4.2) has already been downloaded in the virtual machine for you to use. There is a plugin for Eclipse that makes it simple to create a new Hadoop project and execute Hadoop jobs, hadoop-eclipse-plugin-2.7.2.jar, which is also downloaded. In this exercise, you will learn how to use Eclipse to create a MapReduce project, configure the project, and run the program. You can also manage the files in HDFS by using Eclipse, instead of using commands to transfer files between local file systems and HDFS.

1. Configure the eclipse Hadoop plugin:

a) Open Eclipse, and make the workspace folder at "/home/comp9313/workspace" by default. In "Project Explorer" you will see "DFS Locations":



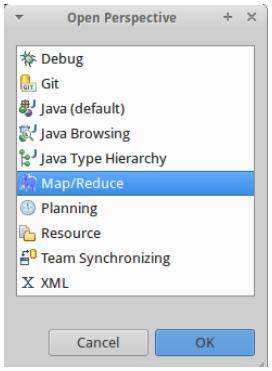
b) In Eclipse Menu, select Window->Preferences, then a dialog will pop up like below:



Configure your Hadoop installation directory as shown in the figure.

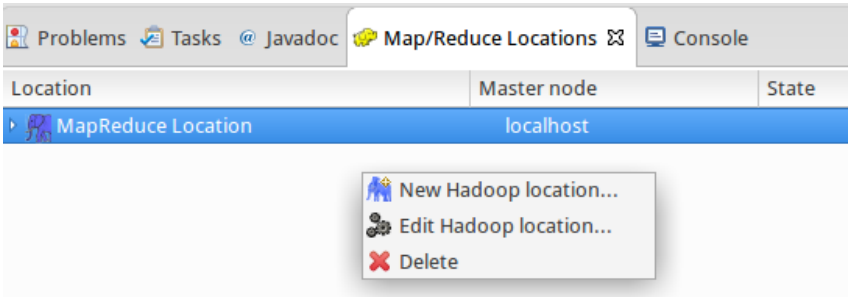
c) Change to the Map/Reduce Perspective:

Select Window->Open Perspective->Other->Map/Reduce

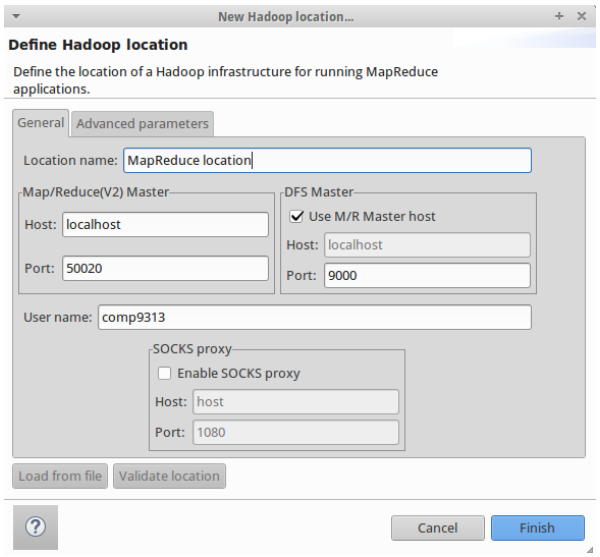


d) Connect Eclipse with HDFS

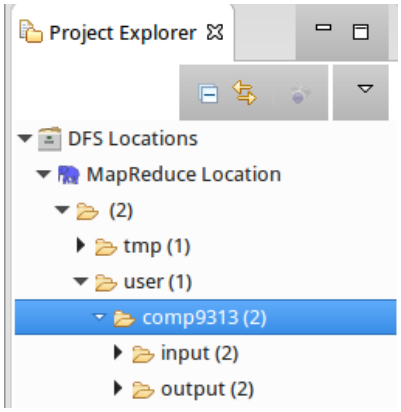
Right click in tab Map/Reduce Locations, and select “New Hadoop location”



In the pop-up dialog, give a name for the Map/Reduce location, and change the port of DFS Master to “9000”



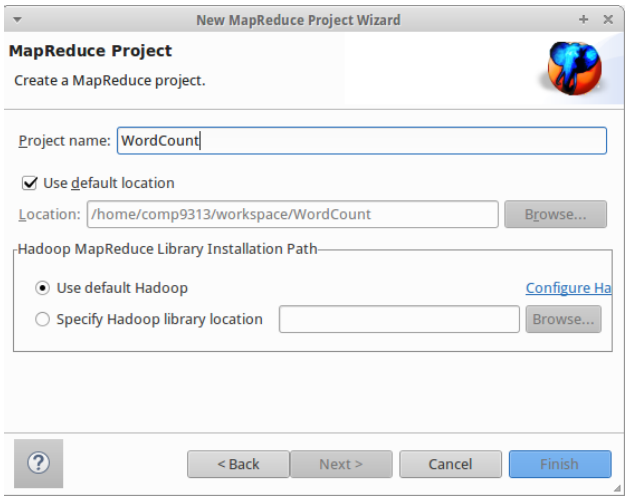
e) Test the connection. If you have successfully connected Eclipse and Hadoop, you can see the folders and files in HDFS under “DFS Locations”.



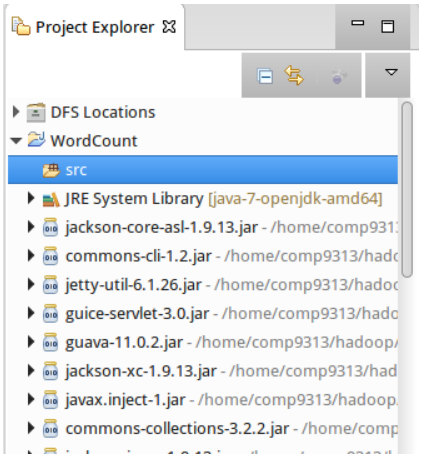
You can click the files to view them, and you can also download files to local file system or upload files to HDFS.

2. Create your WordCount Project in Eclipse

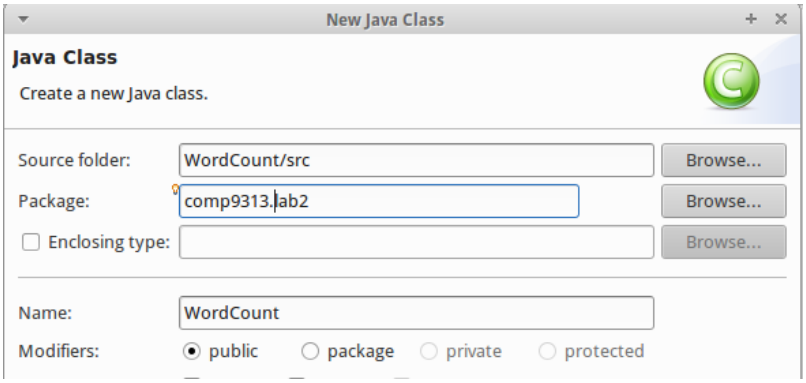
a) Select File->New->Project to create a Map/Reduce project. Name the project as “WordCount”.



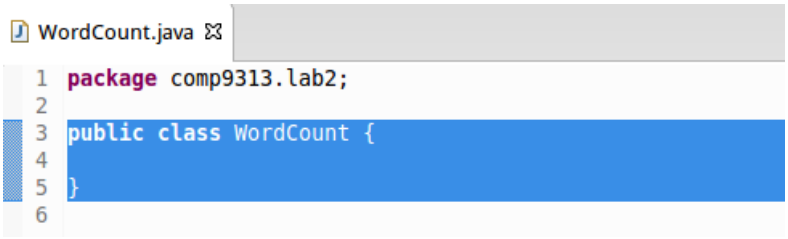
Now you can see the created project in “Project Explorer”.



b) Create a new class “WordCount”, in package “comp9313.lab2”



c) Replace the code of class WordCount by the content of “WordCount.java” in the first exercise.



d) Copy the file "log4j.properties" from $HADOOP_CONF_DIR to the src folder of project "WordCount"

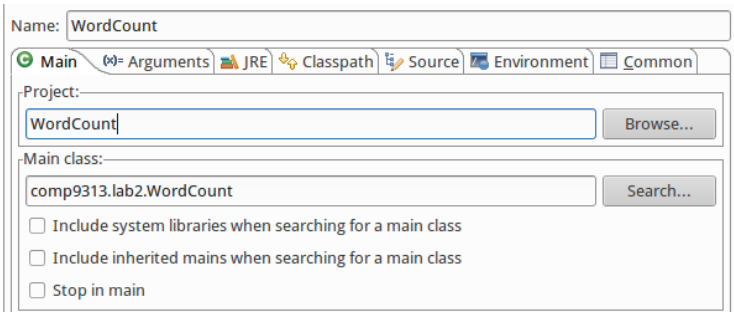
```
$ cp $HADOOP_CONF_DIR/ log4j.properties ~/workspace/WordCount/src
```

Then right click the project in Eclipse and click "Refresh".

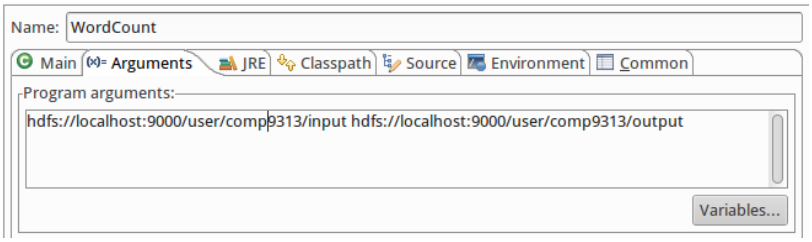
This step is to configure the log4j system for Hadoop. Without doing this, you cannot see the Hadoop running message in Eclipse console.

Running MapReduce Jobs in Eclipse

Right click the new created file WordCount.java, and select Run as->Run Configurations->Java Application. In the dialog, click the tab "Main", and make input "comp9313.lab2.WordCount" as the "Main class".



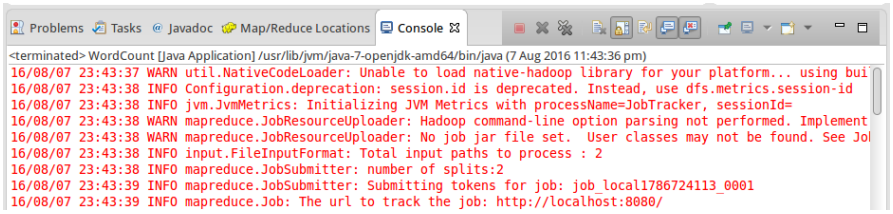
Then configure the arguments for this project: make the arguments as "hdfs://localhost:9000/user/comp9313/input hdfs://localhost:9000/user/comp9313/output". Finally, click "Run".



Warning: Note that if output already exists, you will meet an exception. Remember to delete `output` on HDFS:

```
$ $HADOOP_HOME/bin/hdfs dfs -rm -r output
```

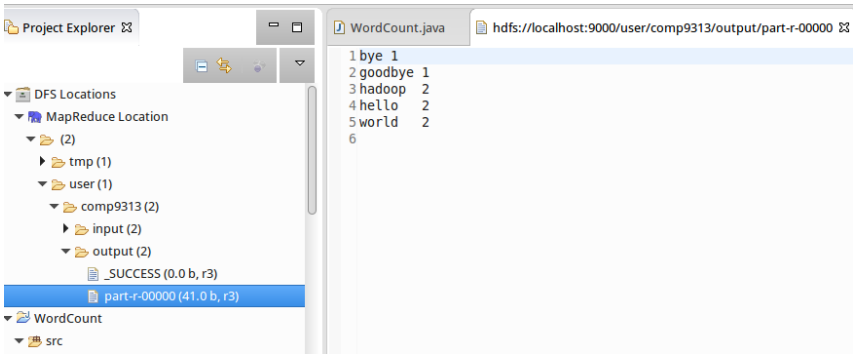
If everything works normally, you will see the Hadoop running message in Eclipse console:



Note: If you still see the following warnings after you run the program, you may need to restart eclipse.

```
log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Refresh “DFS Location”, you will see that a new folder “output” is listed, and you can click the file in the folder to see the results.



Quiz: Split the code into three files: one for mapper, one for reducer, and one for main (driver), and run the project again. Normally, in a MapReduce project, we will put the three classes into different files.

Note that the mapper and reducer classes are not static in this case!

After you have set up the run configuration the first time, you can skip the step of configuring the arguments in subsequent runs, unless you need to change the arguments.

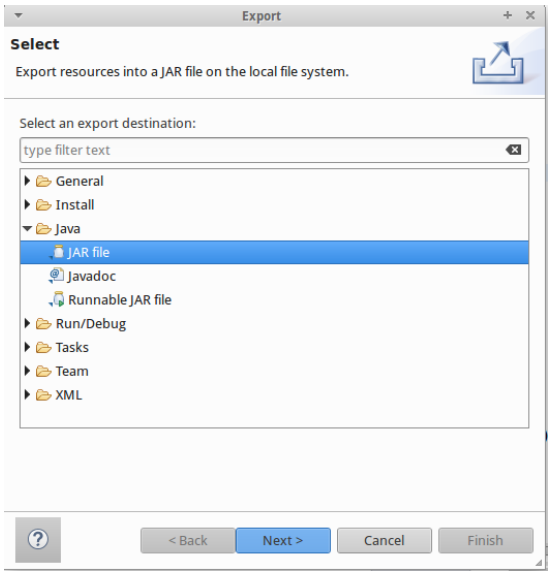
Now you’ve make the MapReduce job run in Eclipse. Note that Eclipse does not use YARN to manage resources.

Package MapReduce Jobs using Eclipse

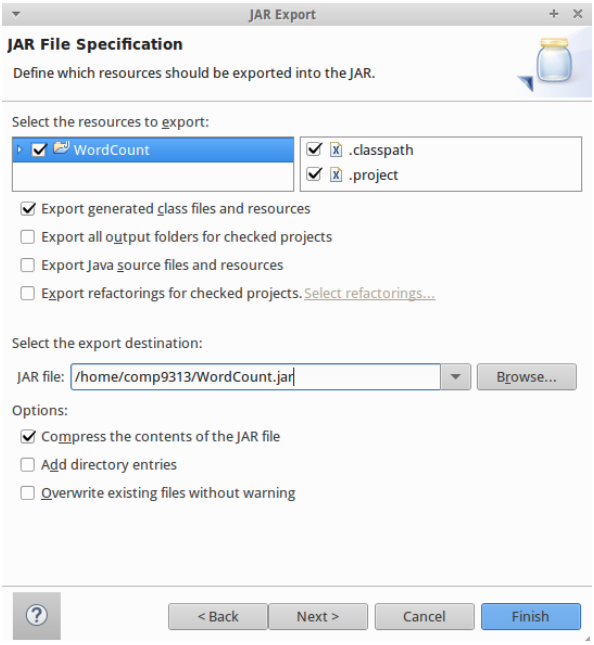
Once you've created your project and written the source code, to run the project in pseudo-distributed mode and let YARN manage resources, we need to export the project as a jar in Eclipse:

1. Right-click on the project and select Export.

2. In the pop-up dialog, expand the Java node and select JAR file. Click Next.



3. Enter a path in the JAR file field and click Finish.



4. Open a terminal and run the following command:

```
$ $HADOOP_HOME/bin/hadoop jar ~/WordCount.jar comp9313.lab2.WordCount
hdfs://localhost:9000/user/comp9313/input
hdfs://localhost:9000/user/comp9313/output
```

Remember to delete the output folder in HDFS first!

You can also simply run the following command:

```
$ $HADOOP_HOME/bin/hadoop jar ~/WordCount.jar comp9313.lab2.WordCount
input output
```

By using the “hadoop” command, I/O is based on the distributed file system by default, and /user/comp9313 is the default working folder.

Debugging Hadoop Jobs

To debug an issue with a job, the easiest approach is to run the job in Eclipse and use a debugger. To debug your job, do the following step.

1. Set a watch point in TokenizerMapper in the while loop:

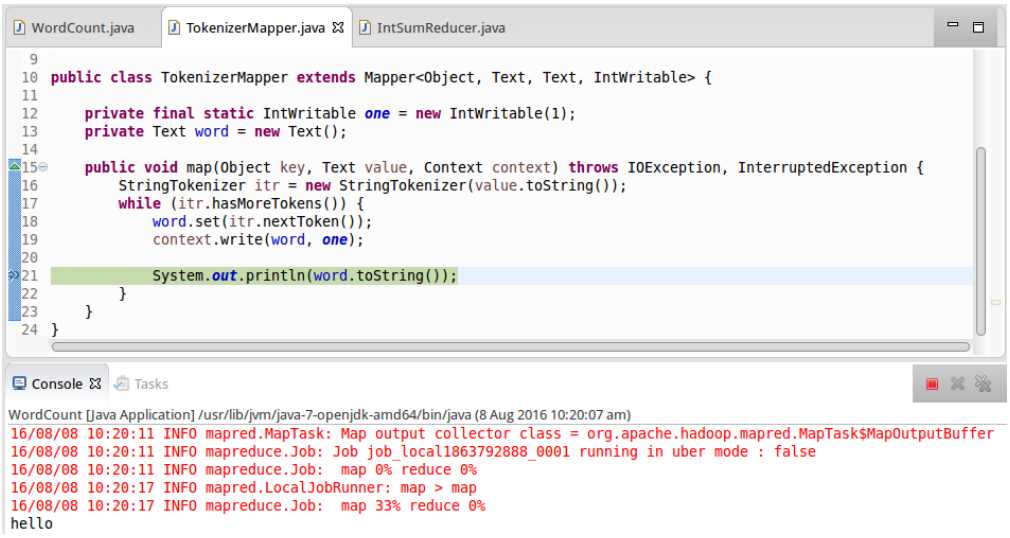
```
while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);

        System.out.println(word.toString());
}
```

Double click the line number of the red line in Eclipse to set the watch point.

2. Right-click on the project and select Debug As -> Java Application, and open the debug perspective.

3. The program will run, and stop at the watch point:



Now you can use the Eclipse debugging features to debug your job execution.

4. Logs are also very useful for you to debug your MapReduce program.

You can either print the debug information in stdout, or write the debug information in the Hadoop system log.

Import the relevant log classes in the java file:

```
import org.apache.htrace.commons.logging.Log;
import org.apache.htrace.commons.logging.LogFactory;
```

In TokenizerMapper, add the following two lines after "System.out.println(word.toString());":

```
Log log = LogFactory.getLog(TokenizerMapper.class);
log.info("MyLog@Mapper: " + word.toString());
```

In the reducer class IntSumReducer, add the following lines at the end of the reduce function:

```
System.out.println(key.toString()+ " " + result.toString());

Log log = LogFactory.getLog(IntSumReducer.class);
log.info("MyLog@Reducer: " + key.toString() + " " +
result.toString());
```

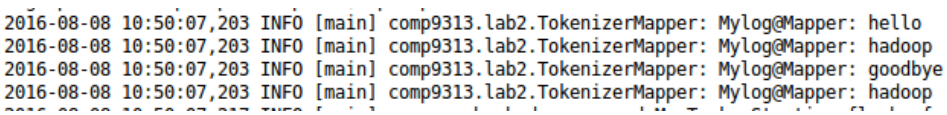
Export the project as a jar file, and run it in the terminal again.

You will find your log messages in logs through different ways:

a) Through http://localhost:50070

Select Utilities->Logs, then click "userlogs/", the log folder of your recent job is shown at the bottom. Go into the folder, and you will see another four log folders.

Each map and reduce will record their own log. Enter the folder ending with "000002", and then click syslog, you can find:



If you click stdout, you can find:

```
hello
hadoop
goodbye
hadoop
```

As you can see, System.out.println() prints the information to stdout, while, the Log class writes the information to syslog.

Enter the folder ending with “000003”, and then click syslog, you can find:

```
2016-08-08 10:50:07,225 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Mapper: hello
2016-08-08 10:50:07,225 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Mapper: world
2016-08-08 10:50:07,226 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Mapper: bye
2016-08-08 10:50:07,226 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Mapper: world
```

Enter the folder ending with “000004”, and then click syslog, you can find:

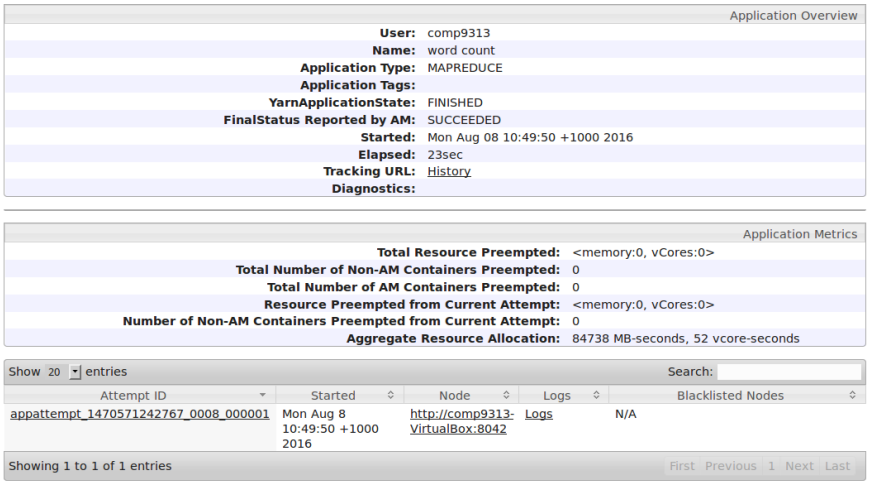
```
2016-08-08 15:19:12,883 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Reducer: bye 1
2016-08-08 15:19:12,883 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Reducer: goodbye 1
2016-08-08 15:19:12,883 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Reducer: hadoop 2
2016-08-08 15:19:12,884 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Reducer: hello 2
2016-08-08 15:19:12,884 INFO [main] comp9313.lab2.TokenizerMapper: Mylog@Reducer: world 2
```

If you click stdout, you will see:

```
bye 1
goodbye 1
hadoop 2
hello 2
world 2
```

b) Through http://localhost:8088

Your recent MapReduce job is listed at the top of the list. Click the application ID, and you will see:



Click Logs, and you can view the logs in the webpage. Note that only the log folder ending with “000001” is shown (i.e., the logs of the driver). You

can change the URL to see other log folders. For example, you can replace "000001" with "000002" to see the logs of the first mapper.

c) Through your local machine.

Open terminal, cd to the Hadoop log folder to check the logs for your job:

```
$ cd $HADOOP_LOG_DIR/userlogs
```

For large MapReduce project, using logs is the best way to debug your code.

Write Your Own Hadoop Job

1. Download the test file, and put it to HDFS:

```
$ wget
https://webcms3.cse.unsw.edu.au/static/uploads/course/COMP9313/18s2/3
3c7707c8b646a686e33af7e2f2fc006b53ff8c13d8317976bd262d8c6daae66/pg100
.txt
$ $HADOOP_HOME/bin/hdfs dfs -rm input/*
$ $HADOOP_HOME/bin/hdfs dfs -put ~/pg100.txt input
```

2. Run the word count java program to check the results.

3. Now please write your first MapReduce job to accomplish the following task:

Write a Hadoop MapReduce program which outputs the number of words that start with each letter. This means that for every letter we want to count the total number of words that start with that letter. In your implementation ignore the letter case, i.e., consider all words as lower case. You can ignore all non-alphabetic characters. Create a class "LetterCount.java" in package "comp9313.lab2 " to finish this task.

Hint: In the (key, value) output, each letter is the key, and its count is the value.

1. How to set a reducer properly?
2. How to write a combiner?

Compare your results with the answer provided at:

https://webcms3.cse.unsw.edu.au/COMP9313/18s2/resources/17731

(**Optional Problem**) Try to work on the following problem: compute the average length of words starting with each letter. This means that for every

letter, we want to compute the total length of all words that start with that letter divided by the total number of words that start with that letter.